Indexing Product Reviews

Overview

Write a program that creates a basic index of customer reviews of shoes and, once finished, allows the user to search these reviews by keyword.

We have a couple thousand text files on a server that we need to make sense of. The first line of each file is a shoe's brand and model name (e.g. Nike Shox Turbo+ 6). The rest of the lines in each file are reviews about the product from customers who've purchased it (one review per line).

The URLs for all of these "review files" are specified in another text file called reviewUrls.txt, one URL per line.

Your program will read the URLs from reviewUrls.txt on the command line, download the review files, build a search index, ask the user to input a keyword, and return the names of all the products containing that keyword.

Example Output

```
...
...
indexing http://s3.amazonaws.com/shoefitr-recruit/review-indexer/AA23C5EDD7C062232F97E8D3F349F167.txt
indexing http://s3.amazonaws.com/shoefitr-recruit/review-indexer/F889F883382028D026038D782664B063.txt
Finished.

Enter a keyword to search: love

in 10 reviews for Onitsuka Tiger by Asics Mexico 66??
in 8 reviews for Brooks Launch
in 7 reviews for Globe Sabre
in 7 reviews for inov-8 F-Lite??? 230
in 7 reviews for inov-8 F-Lite??? 195
in 6 reviews for Chaco Z/2?? Unaweep
in 6 reviews for Globe Fusion
in 6 reviews for Crocs Blitzen Polar
in 6 reviews for adidas Originals Gazelle
in 6 reviews for Onitsuka Tiger by Asics Serrano???
```

Note that this is just sample output – it's not correct for the keyword "love".

The program should enable the user to search many times once the index is created. Try to minimize the amount of time the user has to wait before they can start searching the first time the program is run. Once the index has been created and the user starts searching, the results for each search should be returned "instantly". The solution should scale for a data set of hundreds of reviews to a data set of tens of thousands of reviews.

Tasks

We will be reviewing the code you produce—complete or incomplete—so please add commentary in any places you think we would benefit from further explanation.

1. Design your program to work synchronously (i.e. using only the main thread) and complete the implementation. Make sure the program works as intended. The input/output need not be pretty; we're only interested in the actual search results.
    - When you believe your implementation is in working condition, please save a copy of your source files to a folder called "serial".
2. Rework your program to leverage concurrency (i.e. multiple threads) to speed up the indexing process. The solution need not be complicated or elaborate. What will give you the biggest bang for your buck? How would your program perform for thousands of review files instead of just hundreds? What about hundreds of thousands?
    - When you believe your implementation is in working condition and satisfactory, please save a copy of your source files to a folder called "parallel".

What You Should Focus On

We're not trying to test your ability to create an exotic index to support full-text search. Your program only needs to support searches for single, complete, whitespace-separated, alpha-numeric keywords. We really just want to see that you understand where latency exists and that you have techniques to mitigate that latency when necessary. We're also interested in how you organize your code.

Provided Files

- **reviewUrls.txt**
  A list of URLs, one per line.
- **stopWords.txt**
  A list of very common words, one per line, that you may wish to exclude from indexing.

Questions?

Feel free to e-mail us if you have any questions. We may politely decline to answer design or implementation questions.